GEOMETRIC MODELING USING SUPERQUADRICS

A Thesis Submitted

In Partial Fulfilment of the Requirements

for the Degree of

MASTER OF TECHNOLOGY

by

ASHISH D. KALEY

to the

DEPARTMENT OF MECHANICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

FEBRUARY, 1987

DEDICATED

TO

MY PARENTS

CERTIFICATE

This is to certify that the work entitled "GEOMETRIC MODELING USING SUPERQUADRICS" by ASHISH D. KALEY has been carried out under our supervision and has not been submitted elsewhere for the award of a degree.

Dr. M.K. Muju
Professor
Dept. of Mech. Engg.
I.I.T. Kanpur

9/2/87
Dr. S.G. Dhande
Professor
Dept. of Mech. Engg. and
Comp. Sc. and Engg.
I.I.T. Kanpur

ACKNOWLEDGEMENTS

I take this opportunity to express my profound sense of gratitude to my thesis supervisors Dr. S.G. Dhande and Dr. M.K. Muju for their kind help, guidance and moral support throughout this project.

I would like to thank my pals Ghanti, Ghassu, Inder, Jimmy, George and grandpa who made my stay at I.I.T. Kanpur a memorable one. I must express my gratitude to my friends C.C.S. Reddy and K.G. Shastry for their help at various stages of this work.

Finally, I would like to thank Mr. R.N. Srivastava for his meticulous typing and Mr. J.C. Verma for his excellent drawings.

Ashish Kaley

CONTENTS

LIST OF FIGURES

NOMENCLATURE

A	:	Major axis of elliptic cross section for cylinder and cone
B	:	Minor axis of elliptic cross section for cylinder and cone
C	:	Colour index
D	:	Diffusivity val e
DX	:	X component of direction cosine of the ray
DXN	:	X component of the view normal vector
DY	:	Y component of direction cosine of the ray
DYN	:	Y component of the view normal vector
DZ	:	Z component of direction cosine of the ray
DZN	:	Z component of the view normal vector
E	:	Specularity exponent
H	:	Height of box, wedge, cylinder, cone
I	:	Intensity of the visible surface point
K_d	:	Object light diffusivity
K_s	:	Object light specularity
L	:	Length of the box or wedge
$\hat{L}$	:	Unit light vector
N	:	Superquadric degree
$\hat{N}$	:	Unit normal vector
P	:	Rotation in degrees about Y axis
R	:	Radius of sphere
S	:	Object light specualrity
SX	:	Scaling factor in X direction
SY	:	Scaling factor in Y direction
SZ	:	Scaling factor in Z direction

T	:	Rotation in degrees about Z axis
t	:	Ray parameter
W	:	Rotation in degrees about X axis
X	:	Translation along X axis
XO	:	Starting X coordinate for the ray
XUP	:	X component of the view up vector
XR	:	X coordinates of view reference point
Y	:	Translation along Y axis
YO	:	Starting Y coordinate for the ray
YR	:	Y coordinates of view reference point
YUP	:	Y component of the view up vector
Z	:	Translation along Z axis
ZO	:	Starting Z coordinate for the ray
ZR	:	Z coordinates of view reference point
ZUP	:	Z component of the view up vector
ϕ	:	Angle of rotation about Y axis
θ	:	Angle of rotation about Z axis
w	:	angle of rotation about X axis

ABSTRACT

In the present work, a scheme of geometrically modeling superquadric solid objects using the constructive solid geometry representation has been designed and implemented. A superquadric surface enables the designer to model objects having fillet-radii and rounded edges using a single bi-parametric equation. Also, a variety of different types of objects can be defined by varying the value of the exponent in the bi-parametric equation. Algorithms for the boolean operations of solids defined by superquadric surfaces have been developed using the constructive solid geometry approach.

In order to render the objects in a realistic manner, the ray tracing technique has been used. Also, the shading of surfaces has been achieved by using Phong's shading model. The entire scheme of geometric modeling has been implemented in the form of a menu-driven program on ND-560 system using Tektronix-4109 graphics display system. Illustrative displays of primitives and objects are presented.

CHAPTER 1

INTRODUCTION

1.1 Geometric Modeling - A Concept

Computer graphics has grown as an area of computer applications enormously. It has developed applications in every possible field ranging from arts having use in computer painting and animation, in medical field for CAT scanning. Also, computer graphics has got wide engineering applications in the field of Robotics,CAD, CAM, finite element analysis and geometric modeling to name a few.

Geometric modeler is essentially a computer based system which provides an efficient tool for creating, editing and storing the object representation needed for different engineering applications. Display generation is an integral part of the geometric modeler. An unambiguous display of the solid is an information bank in itself. Majority of the processes which use the modelled geometry to compute the properties of the object (such as mass, moment of inertia, volume etc.) are application subsystems outside the geometric modeler.

Representations in a geometric modeler are through an input language. It may range from simple interactive graphics command language to an elaborated block structured

language similar to conventional programming languages. The sequence of the input language statements is itself an object representation.

1.2 Geometric Definition of Solid

The core of the geometric modeler consists of the symbol structures (representations) designating "Abstract solids" which are manipulated to model the desired solid.

The abstract solid has following characteristics [1]:

i) RIGIDITY : It should have an invariant configuration or shape which is independent of solids location or orientation.

ii) HOMOGENEOUS THREE DIMENSIONALITY : A solid must have an interior. It cannot have isolated or dangling portions.

iii) FINITENESS : A solid must occupy finite portion of the space.

iv) CLOSURE UNDER BOOLEAN OPERATION : Rigid motions or boolean operations when applied to solids must produce other solids. Drilling, welding, assembly and design operations in general can thus be simulated in this way.

v) FINITE DISCRIBABILITY : There must be some finite aspect of E^3 (subset of 3-D Euclidean space, e.g. finite number of faces) to ensure that they are representable in computers.

1.3 Representation Schemes

Traditional way of representation of the solid is by projection of solid, but this is an ambiguous representation unless one gives sufficient views of the same solid to remove ambiguities. The ambiguity arises because of the loss of information that occurs when a 3-D object is represented on 2-D picture plane.

Some of the unambiguous representations are:

a) Pure Primitive Instancing Scheme: Using this scheme one can represent only a limited range of the objects. Its drawback is that it cannot combine the created objects to model complicated objects and it is very difficult to compute properties of the solid represented by this scheme. The advantage of this scheme is that it is concise and easy to use. It is like a language defined by its grammer. However, in this scheme, it is not possible to have explicit commands for boolean operations.

b) Spatial Occupancy Enumeration: Representation of solids in this scheme is essentially a list of cells occupied by a solid. The cells sometimes called "Voxels" are cubes of fixed size which lie on a fixed spatial grid. Each cell may be represented by coordinates of a single point namely the centroid. It is mostly used to represent complex biological objects

encountered in computer aided tomography[9].

Octree encoding is a special case of spatial enumeration scheme. Octree is 3-D extension of 2-D quadtree encoding concept. In this scheme, a volume is divided into 8 octants till each part is either a homogeneous cell or contains a Voxel (Figure 1.1).

c) Constructive Solid Geometry: Constructive solid geometry (C.S.G.) representation of the object is a binary tree whose nonterminal nodes represent operators which may be rigid motion like translation/rotation or regularised set operators like union intersection or difference. Every nonterminal node also represents a solid. The terminal nodes are the building blocks (primitive solids) such as cylinder, box, wedge etc. The basic idea is that complicated solids can be represented as various ordered additions and substractions of some simpler solids or primitives. In this approach, the modelled object is stored as combination of data and logical procedures (boolean operation). This generally requires less storage but more computation to reproduce the model. Figure 1.2 illustrates the C.S.G. representation [1].

d) Boundary Representation: In this scheme, a solid is represented by segmenting its boundary into finite number of bounded subsets called faces, edges,

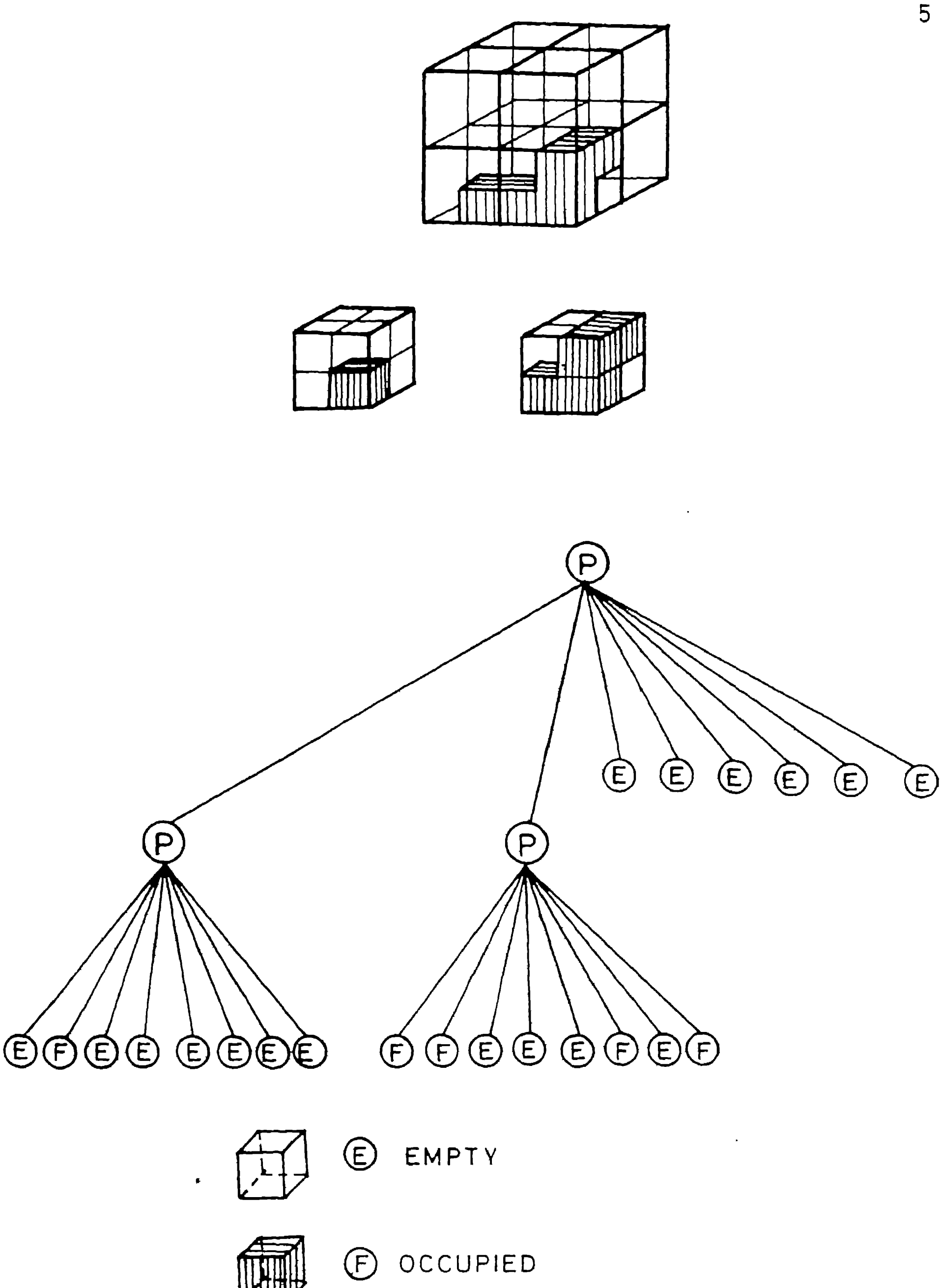


FIG.1·1 OCTREE REPRESENTATION OF THE OBJECT

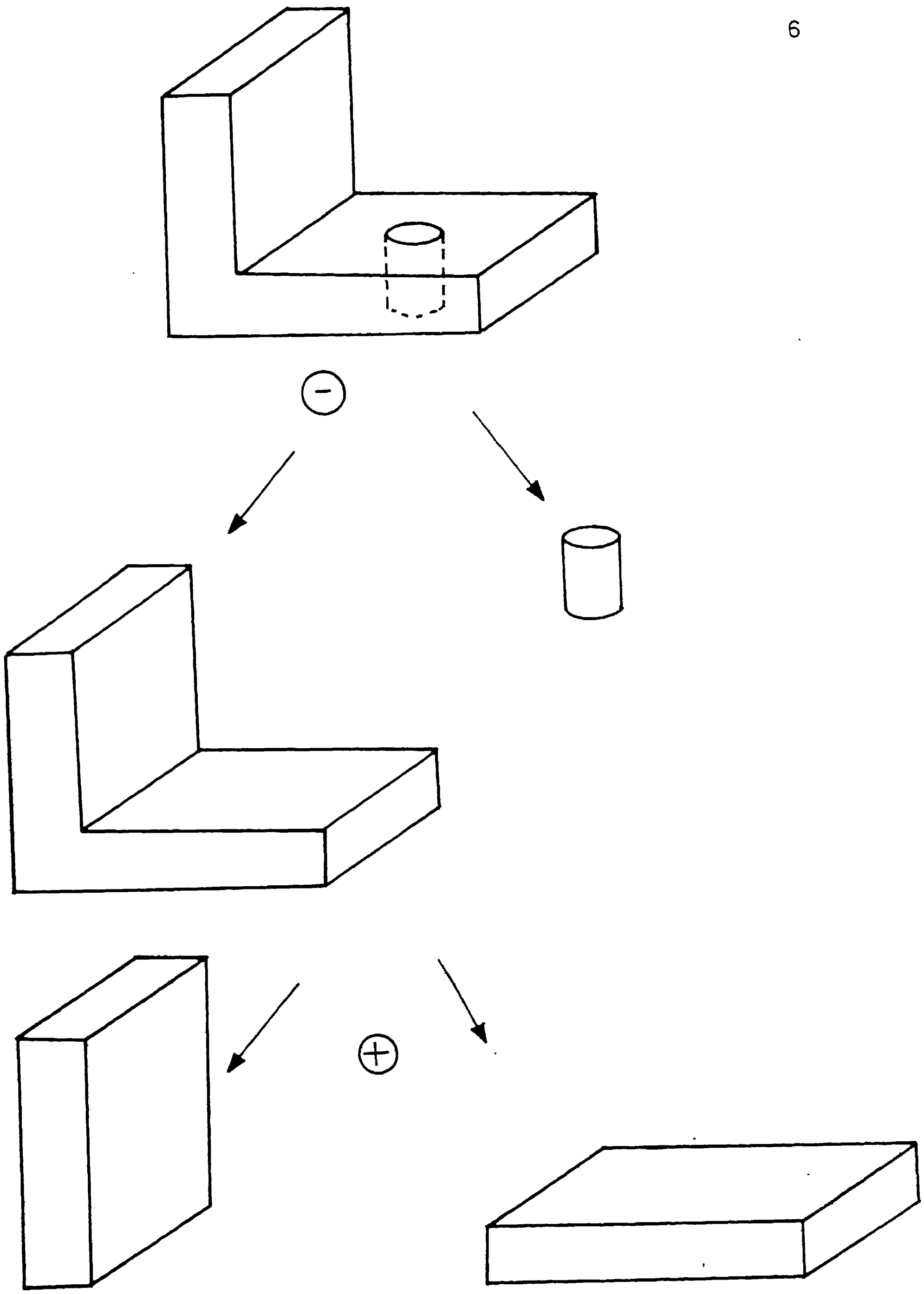


FIG. 1.2 CONSTRUCTIVE SOLID GEOMETRY REPRESENTATION

vertices. Boundary representation scheme is unambiguous if the faces are represented unambiguously [2].

Following are the combinatorial conditions for true boundary representation:

i) Each face must have atleast three edges.

ii) Each edge must have precisely two vertices.

iii) Each edge must belong to even number of faces.

iv) Each vertex in a face must belong precisely to the two of the face's edges.

In boundary representation all the information is explicitly stored by defining the model geometry (faces, edges, vertices etc.). This requires more storage space but comparatively less computational effort to construct the image. This is helpful in modeling very complex geometries (Figure 1.3).

e) Sweep Representation [2]: In this scheme a solid object is represented by moving a set along a trajectory in space which sweeps a volume. There are two types of sweep operations. Translational sweep and Rotational sweep (Figure 1.4). Simple sweep representations are easy to construct. But the domain of the objectsthey can describe contains only those objects that exhibit translational or rotational symmetry. Various generalised forms of sweeping are potentially capable of covering much large domains,

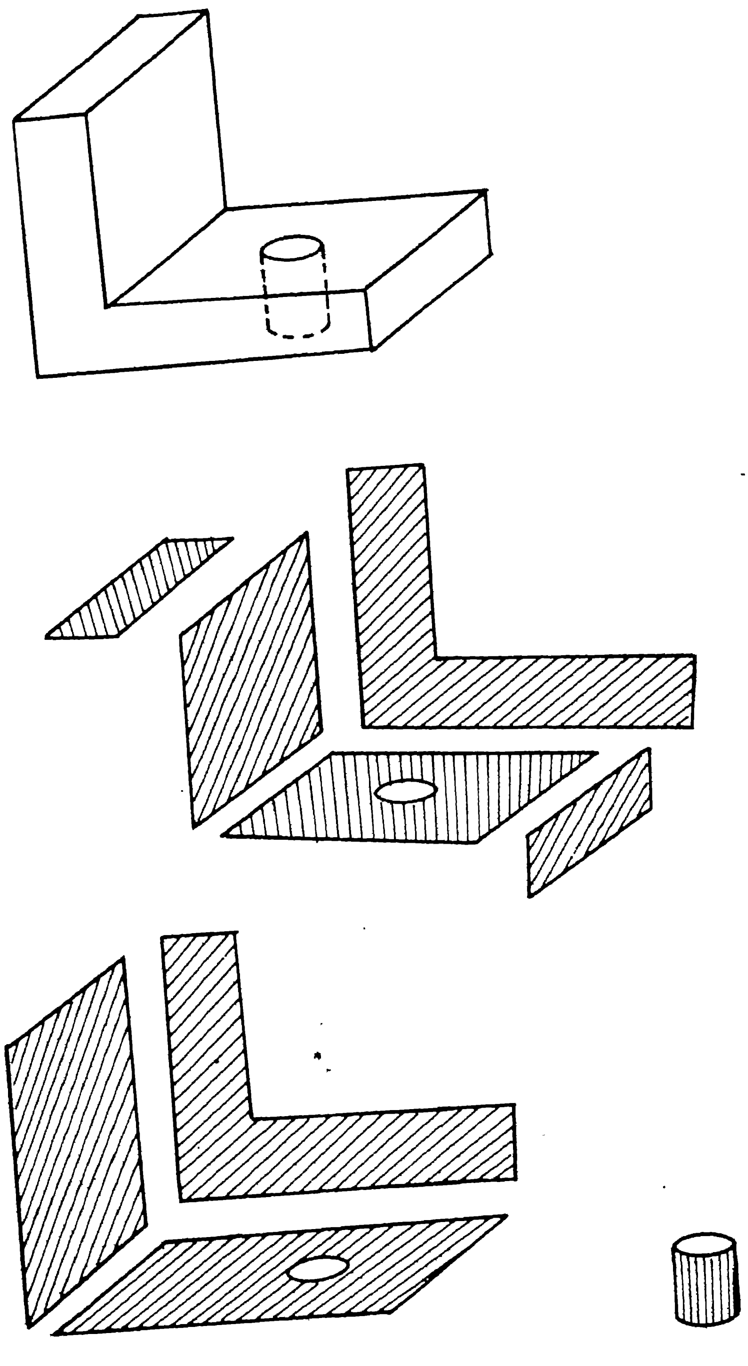
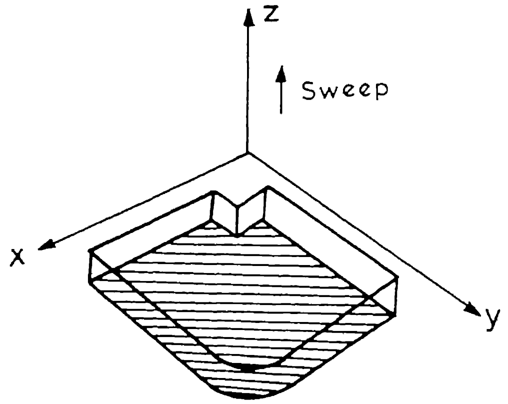
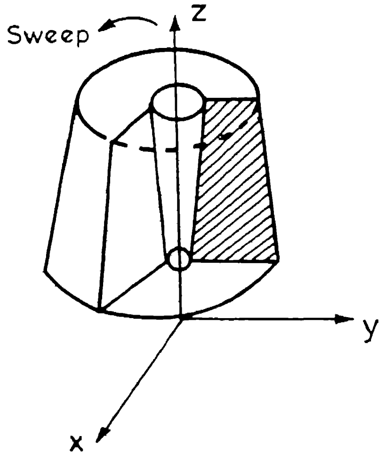


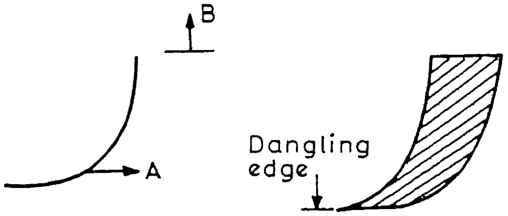
FIG. 1.3 BOUNDARY REPRESENTATION



Translational sweep



Rotational sweep



Generalised sweep producing undesired object

FIG.1·4 VARIOUS SWEEP REPRESENTATIONS

but they pose various mathematical and computational issues. Generalised sweeping sometimes creates unwanted objects with dangling edges or faces.

1.4 Statement of the Problem [3]

One can create all the objects to be designed by manipulating a small library of fundamental solids (primitives) whose properties are well understood.

Most of the solid modelers like GMSOLID, PADL, EUKLID, TIPS [1] use quadric solids such as cylinders, spheres, cones, alongwith boxes and wedges.

These primitives cover a restricted range of solids and do not give the designer sufficient flexibility for modeling.

There is, however, a new family of parametric objects called 'superquadrics'. These are generalizations of conic sections with the exponent 2 replaced by an appropriate positive number, n. These are flexible enough to represent large families of shapes which can be rounded, square etc. These can have different properties in different directions. The intent in the present work is to develop a geometric modeling scheme using 'superquadrics'.

To understand the concept, consider the following equations,

$$x = a \cos^{2/N}\beta \qquad 0 \leq \beta \leq 360 \qquad (1.1)$$
$$y = a \sin^{2/N}\beta$$

Figure 1.5 shows how N controls the shape of the curve. For N = 2, it is a circle and as N approaches ∞ the curve becomes squarish.

This interesting behaviour was the source of motivation for taking up the present work. In the present thesis, geometric modeling system based on constructive solid geometry scheme has been developed. The primitives used are superquadric solids replacing the generally used quadric solids.

1.5 Organisation of Present Work

Chapter 2 gives brief summary of the Ray tracing theory, camera model for viewing, different primitives, coordinate systems, and the shading model.

Chapter 3 deals with the boolean operations, algorithms for creating an object, boolean operation, Ray object intersection and data structure used for representing primitives and solids.

Chapter 4 gives the basic structure of the program developed, different interactive features possible, procedures to be followed for creating/editing the modelled object, graphic displays and implementation details.

Chapter 5 gives summary and scope for further work.

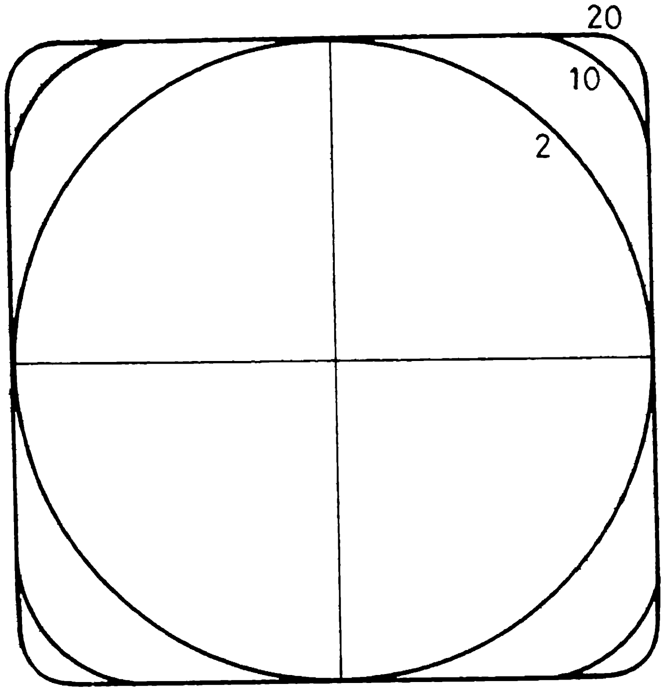


FIG.1·5 SUPER ELLIPSES $X^N + Y^N = 1$ FOR DIFFERENT N

CHAPTER 2

RAY TRACING

2.1 Ray Tracing Principle

Ray tracing is essentially a photographic process in reverse. For each pixel (picture cell) element on the screen a light ray is cast into the scene.

This ray acts as a sort of tracer. By tracing along the ray, ray-object intersections are calculated. These intersection points form the database which is used for modeling the objects. Modeling is achieved through boolean operations. Generation of shaded pictures or line drawings, calculation of various properties of the objects (such as volume, mass, moment of inertia) become a simple affair even for complicated objects.

2.2 Camera Model [7]

For viewing an object in any desired configuration, there are two equivalent ways of going about this:

i) One can think of the view plane (in this case, the screen of the display device) to be fixed and object rotated for the desired configuration.

ii) By keeping the object fixed one can position the view plane according to the requirement (camera model).

For the present work second approach is used. This reduces the mathematical complexity involved in Ray-solid intersection procedure.

Various viewing parameters used are:

a) <u>View Reference Point</u>: This is the most important parameter in the camera model. It is a point in the scene which decides the portion of the scene that will be displayed on the screen. It acts as a sort of reference as all other parameters are taken with reference to this only.

b) <u>View Plane Normal Vector</u>: View plane normal vector gives the direction perpendicular to the view plane directed towards the object. This means that the camera is always along the normal towards view reference point.

c) <u>View Distance</u>: It is the distance between the view reference point and the view plane measured along the view plane normal vector.

d) <u>View Up Direction</u>: This parameter fixes the camera angle in the view plane. By changing this parameter one can imagine the camera pointing at the same object from the same direction but the picture will be tilted accordingly (Figure 2.2).

By changing these parameters one can select different views of the object (Figure 2.1).

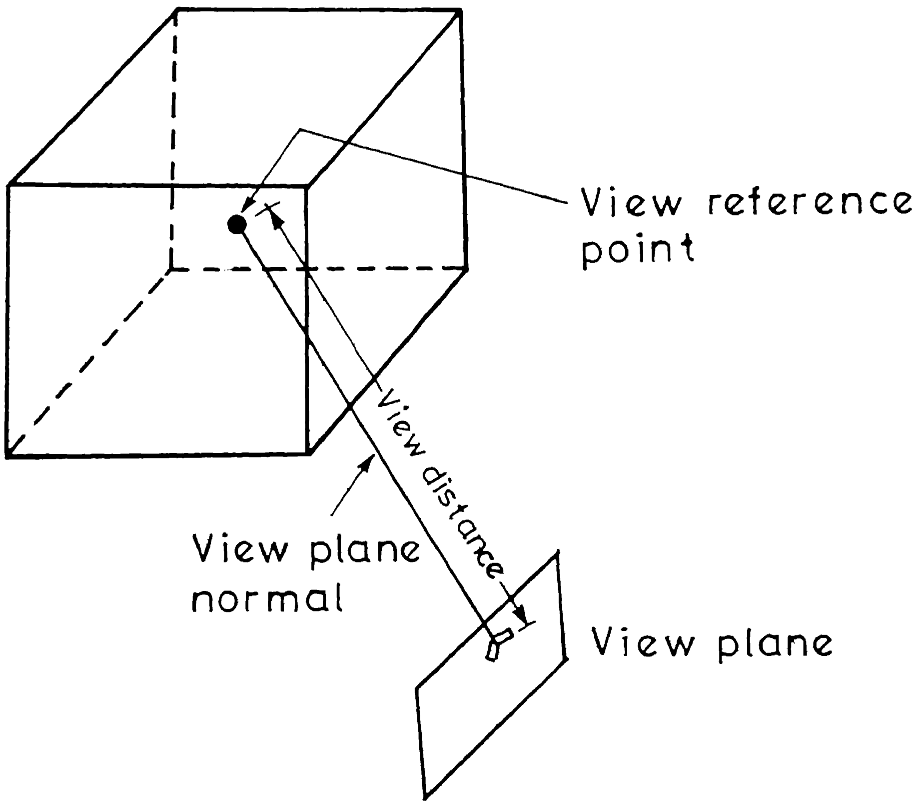


FIG.2.1 DIFFERENT VIEWING PARAMETERS

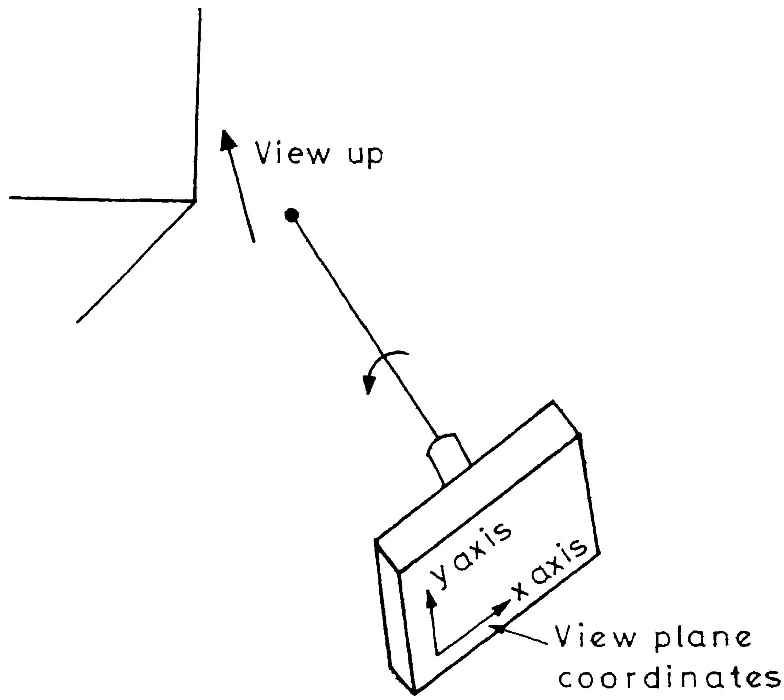


FIG. 2·2 EFFECT OF VIEWUP VECTOR

2.3 Coordinate Systems and Transformations

There are three different coordinate systems used for ray tracing technique. These are:

i) Global Coordinate System: This is the most important coordinate system because in this coordinate system, user positions the camera to select a particular view, positions source of light for shading the object. User creates the object, performs boolean operations and modifies the object - all in this coordinate system only.

ii) Primitive's Local Coordinate System: This coordinate system is primarily used to simplify the geometric calculations involved in ray-solid (primitive) intersection.

Each primitive has a local coordinate system. Whenever a user creates new primitive solid, its position and scaling in the global coordinates are same as those in the local coordinates. But when a user moves or scales it then its position or size changes in the global coordinate system only. 'Local-to-global-transformation-matrix' (Appendix A) is the link between the two coordinate systems.

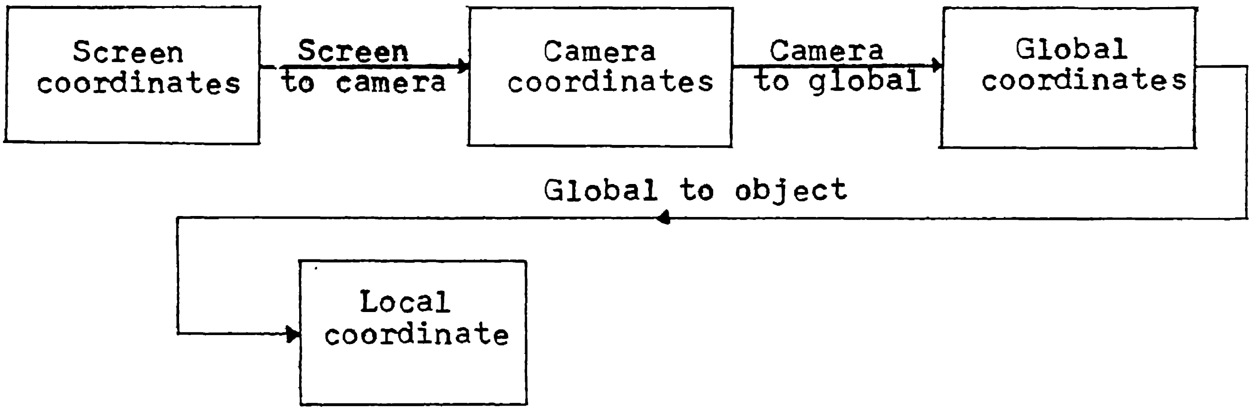
Ray-solid intersection can be found out in primitive's local coordinates and subsequently the intersection point can be transformed to global coordinates.

iii) Screen Coordinate System: Generally the screen or

the view plane is in the X-Y plane, the scene (object) is in the +Z half space and the viewer is in the -Z half space.

The ray is generated in the screen coordinates. It is transformed to primitive's local coordinates, through certain in-between transformations for the calculation of ray-solid intersection points.

The block diagram for various transformations is given as



As parallel projection is used in this work, the direction of ray in the global coordinate is along view normal camera-to-global-transformation matrix is given in Appendix A. Global-to-object-transformation matrix is given in Appendix A.

2.4 Ray Primitive Intersection [5]

A ray is a simple 3-D line in the camera model. It can be best defined in the parametric form as a point

P(XO, YO, ZO) and a direction vector V(DX, DY, DZ). In this form points on the line are ordered and accessed by a single parameter t. For every value of t the corresponding point on the line is given by

$$x = XO + t\ DX$$

$$y = YO + t\ DY \tag{2.1}$$

$$z = ZO + t\ DZ$$

For transforming the ray from one coordinate system to another, only the point and vector transformations are essential.

If [T] is 4 x 4 linear transformation matrix then

$$P'(X', Y', Z', 1) = P(X, Y, Z, 1) * [T] \tag{2.2}$$

Similarly for the direction vector

$$V'(DX', DY', DZ', 1) = (DX, DY, DZ, 1) * [T] \tag{2.3}$$

So, a line is simply transformed by

$$(XO', YO', ZO', 1)(DX', DY', DZ', 1) = ((XO, YO, ZO, 1) * [T])(DX, DY, DZ, 1) * [T])) \tag{2.4}$$

From the above equations it is obvious that the line parameter 't' is same in both the coordinate systems.

This is very important because ray-primitive intersection procedure finds only the parameter 't'. For the calculation of intersection points solid or surface equations are not transformed, instead all the transformations are done on the ray. The effect of scaling, rotating and translating a solid is achieved by performing the above operations only on the ray. This simplifies the ray-primitive intersection calculation to great extent.

The block diagram for transformation is given in Section 2.3.

2.5 Primitive Definition [5]

As this work is based on C.S.G. representation, there are certain basic building blocks or primitives used which satisfy all the criteria of the "Abstract Solid" given in Section 1.2.

The primitives used are:

a) Box: It is a unit cube in the positive octant with one corner at origin (Figure 2.3).

The surface equations and the bound tests are

$$X = 0 \qquad 0 \leq Y, Z \leq 1$$

$$X = 1 \qquad 0 \leq Y, Z \leq 1$$

$$Y = 0 \qquad 0 \leq X, Z \leq 1$$

$$Y = 1 \qquad 0 \leq X, Z \leq 1$$

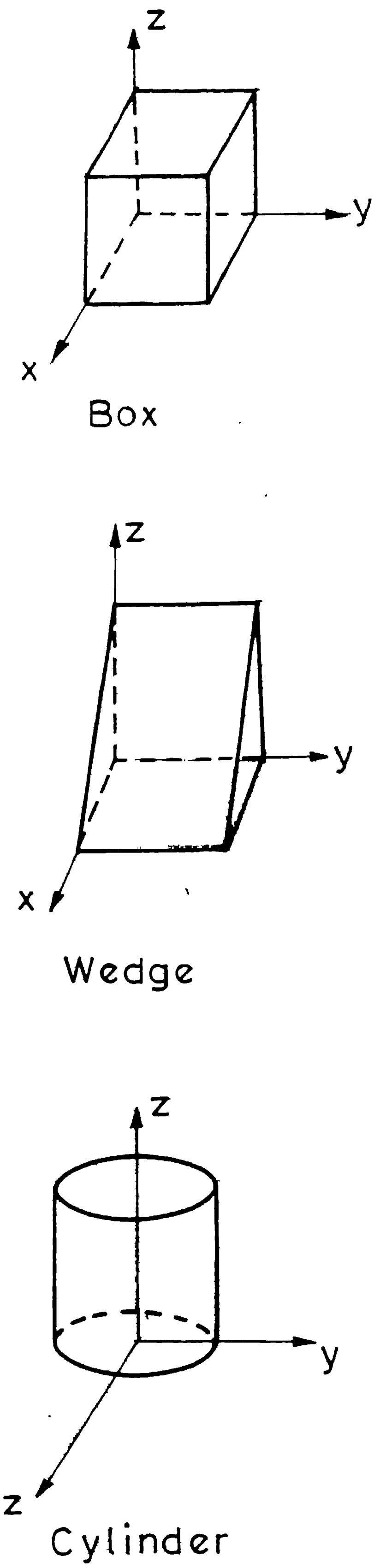
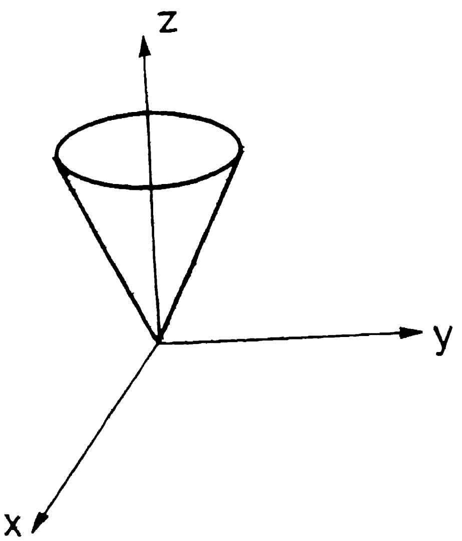
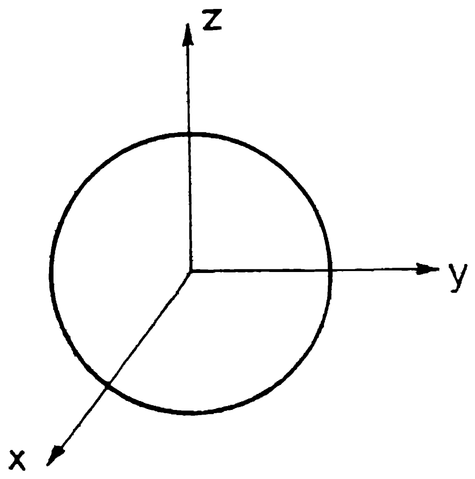


FIG. 2·3 PRIMITIVES IN LOCAL COORDINATE CONFIGURATION



Cone



Sphere

FIG. 2.3 CONTD.

$$Z = 0 \qquad 0 \leq X, Y \leq 1$$

$$Z = 1 \qquad 0 \leq X, Y \leq 1$$

To get the ray-box intersection we have to perform 6 ray-plane tests.

b) Wedge: The wedge is shown in the Figure 2.3. It has length, breadth and height equal to unity.

The surface equations and the bound tests are

$$X = 0 \qquad 0 \leq Y, Z \leq 1$$

$$X + Z = 1 \qquad 0 \leq Y, Z \leq 1$$

$$Y = 0 \qquad \begin{array}{l} X \geq 0 \\ X + Z \leq 1 \\ Z \geq 0 \end{array}$$

$$Y = 1 \qquad \begin{array}{l} X \geq 0 \\ X + Z \leq 1 \\ Z \geq 0 \end{array}$$

$$Z = 0 \qquad 0 \leq X, Y \leq 1$$

To get ray-wedge intersection points we have to perform 5 ray-plane intersection tests.

c) Cylinder: It is a superquadric cylinder with its axis along the positive Z axis with one end at the origin (Figure 2.3).

The surface equations with the bounds are

$$X^N + Y^N = 1 \qquad 0 \leq Z \leq 1$$

$$Z = 0 \qquad X^N + Y^N \leq 1$$

$$Z = 1 \qquad X^N + Y^N \leq 1$$

To get the intersection points two ray-plane intersection tests and one ray-superquadric surface intersection test, need to be performed.

d) <u>Cone</u>: It is a superquadric cone having its axis along the positive Z direction with the vertex at the origin (Figure 2.3).

The surface equations alongwith the bounds are

$$X^N + Y^N - Z^N = 0 \qquad 0 \leq Z \leq 1$$

$$Z = 1 \qquad X^N + Y^N \leq 1$$

For the ray-cone intersection points we have to perform one ray-plane and one ray-superquadric surface intersection test.

e) <u>Sphere</u>: It is a superquadric sphere having its radius unity and the centre at the origin (Figure 2.3).

The surface equation is

$$X^N + Y^N + Z^N = 1$$

To get ray-sphere intersection we have to perform one ray-superquadric surface intersection test.

2.6 Rendering

Rendering is essentially producing the realistic pictures or images.

In a simple shading model (Lambertian reflection model) [8] a source of light is positioned with respect to the object. For every visible surface point on the object, light intensity is calculated by calculating the angle between the normal and the line joining the surface point and the light source.

Objects rendered with a simple Lambertian diffuse reflection illumination model appear to have dull matte surface because a point light source is assumed. Objects that receive no light directly from the source appear black. However in real scene, objects also receive light scattered back to them from surrounding.

An empirical shading model given by Bui-Tong-Phong [8] is implemented in this work. This model is explained as follows (Figure 2.4).

If I is the intensity of the surface as viewed by the observer then

$$I = K_d * (\hat{L} . \hat{N}) + K_s * (\hat{V} . \hat{N})^E$$

where

K_d : diffused reflection constant

$\hat{N}$: unit surface normal

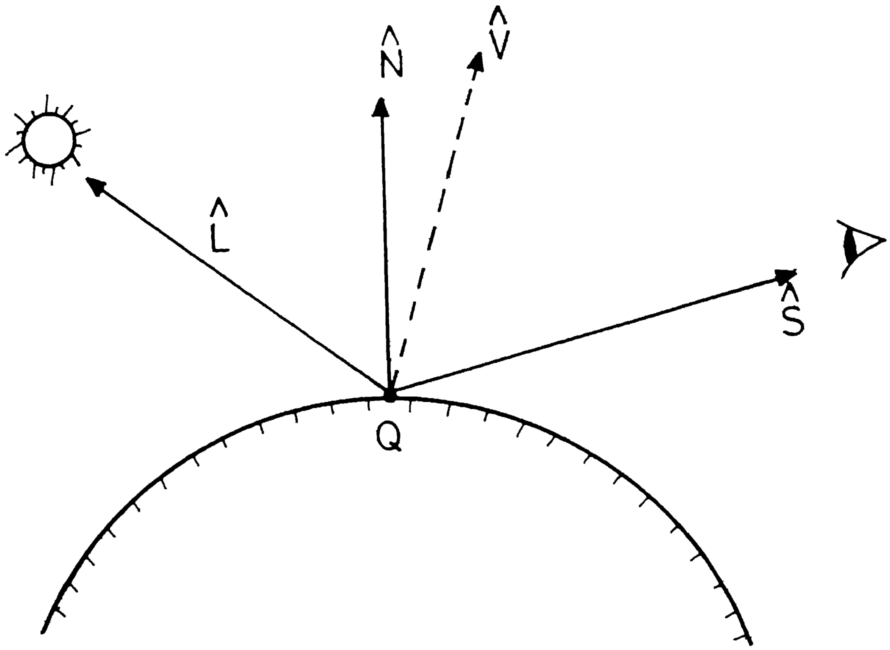


FIG. 2·4 SPECULAR REFLECTION MODEL

K_s : specularity constant

$\hat{L}$: unit light vector

$\hat{V}$: $\hat{L} - \hat{S}$ ($\hat{S}$ - unit sight vector)

E : constant which determines the spatial distribution characteristics of metallic or non-metallic surfaces.

CHAPTER 3

BOOLEAN OPERATIONS AND ALGORITHMS

3.1 In Out Classification [5]

For a given primitive, ray-primitive intersection has four possible outcomes:

1) Ray missing the primitive
2) Ray touching the primitive at one point
3) Ray enters and exits through primitive at two different points
4) Ray lies on face of a primitive.

For cases (1) and (2), a ray is classified as completely out and so no intersection points are noted. In cases (3) and (4), two intersection points are noted and the ray is divided into 3 segments: out-in-out. This classification is very important for carrying out the boolean operations.

3.2 Data Structure

As mentioned earlier the data structure representing the constructive solid geometry scheme is an inverted binary tree.

The terminal nodes of the tree are the different primitives used. The internal nodes are the composite solids formed by different boolean operations.

The data structure used to represent a primitive solid has a parameter attribute. This stores the information

such as superquadric degree, rotation/translation of the primitive about the axes, different dimensions such as length, breadth, height, major and minor radii depending on primitive type.

It also has world-to-object and object-to-world transformation matrices as attributes. These are used for representing the primitive in global coordinates or transforming the ray in primitive's local coordinate system.

The composite solid has operation attribute which specifies the boolean operation UNION (+), DIFFERENCE (-) or INTERSECTION (*). It has left and right pointers for pointing to left subtree and right subtree. The last attribute is leaf number. If this is zero then the node is a terminal node.

All the objects created including the primitive solids are accessed by pointer which points to a particular node of the tree depending upon the name attribute.

3.3 Boolean Operations

Solid primitive and boolean operation comprise a natural and powerful design mode. For example, when a designer wants to drill a hole in a part what could be more natural than to place a cylinder in the desired position of the hole and then substract it from the part.

In a C.S.G. representation solid objects are modelled as composition of primitive solids using boolean operations namely union, intersection and difference.

The ray solid intersection procedure starts at the top of the tree, recursively descends to the bottom classifying the ray with respect to primitive solid and then returns up the tree combining the classification of the left and right subtrees (Figure 3.1). In Figure 3.1 dotted portion of the ray is inside the solid and solid portion is outside the solid.

The boolean operations are simple set operations. The rules are given below:

Operator	Left	Right	Composite
Union	IN	IN	IN
	IN	OUT	IN
	OUT	IN	IN
	OUT	OUT	OUT
Intersection	IN	IN	IN
	IN	OUT	OUT
	OUT	IN	OUT
	OUT	OUT	OUT
Difference	IN	IN	OUT
	IN	OUT	IN
	OUT	IN	OUT
	OUT	OUT	OUT

To explain these operations more clearly consider the example as shown in Figure 3.2. Solid line represent ray portion inside the solid and dotted indicates that ray is outside the solid.

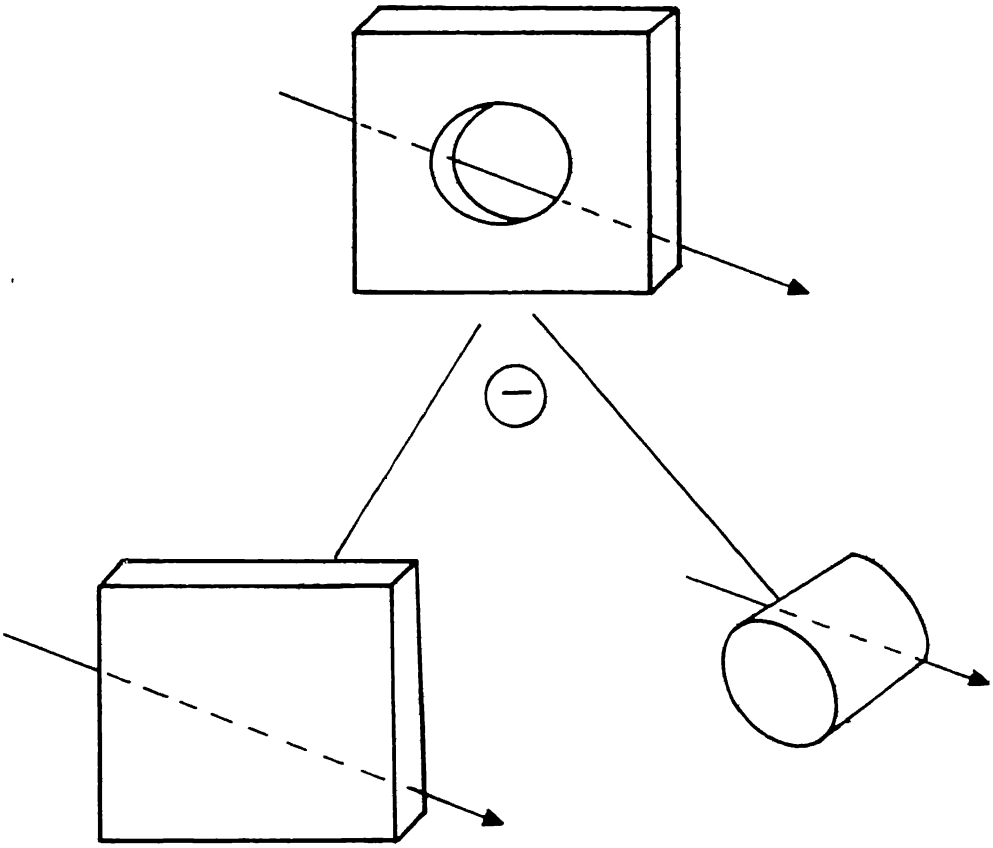


FIG. 3-1 SIMPLE COMPOSITION TREE

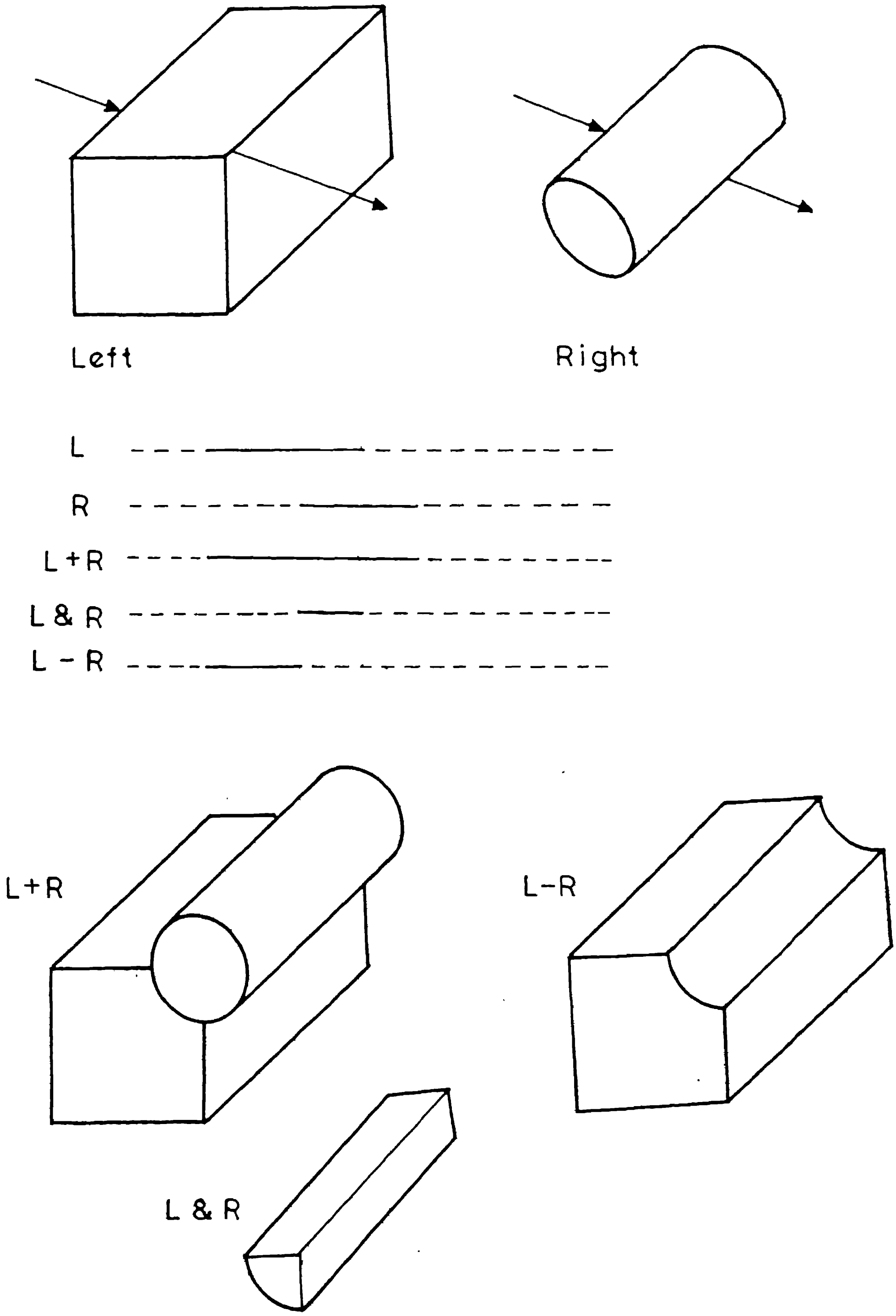


FIG.3.2 BOOLEAN OPERATIONS

3.4 Algorithm for Creating the Object

This procedure as the name suggests when invoked creates a new object. Different cases for creating the new objects are:

1) Two primitives can be combined to form a new object
2) One object (already created) and one primitive can be combined to form a new object
3) Two objects (already created) can be combined to form a new object
4) Already existing object can be modified.

These features give the user the flexibility needed to create new objects.

The algorithm is presented below:

a] Read the object name from TTY and store the object in a stack; get the corresponding stack value.

b] Assign the stack value to a variable LHS.

c] If LHS is less than zero then
   i) prompt error message
      Attempt to redefine primitive
   ii) exit from procedure.

d] Read a character from TTY

e] If character is not equal to '=' then
   i) prompt error message
      Equal to '=' expected
   ii) exit from procedure.

f] Read the object name from TTY and pop the object stack to get the corresponding stack value.

g] Assign the stack value to a variable RH1.

h] If RH1 is equal to zero then

i) prompt message

Undefined object! try again

ii) exit from procedure.

otherwise

if RH1 is less than zero then

i) assign the parameters if any to RH1

ii) create a node with pointer to left

iii) assign the node the corresponding leaf number

otherwise

i) make a copy of the objects of the left subtree

ii) if parameters then modify the object.

i] Read a character from TTY.

j] If character is equal to ';' then

If RH1 is less than zero then

i) create a new node

ii) assign the corresponding leaf number to this node

iii) store the object in the object stack

otherwise

i) store the modified object in the object list

ii) set the pointer to indicate .to left subtree.

k] Read the boolean operation from TTY.

l] Read the object name and get the corresponding stack value for this object.

m] Assign stack value to variable RH2.

n] If RH2 is equal to zero then

 i) prompt message

 Undefined object! try again

 ii) exit from procedure

 otherwise

 If RH2 is less than zero then

 i) assign the parameters if any to RH2

 ii) create a node with pointer pointing to right

 iii) assign the node corresponding leaf number

 otherwise

 i) make a copy of the objects in the right subtree

 ii) if parameters then modify the object.

o] Create a node, assign the left pointer to the left subtree, right pointer to the right subtree.

p] Make an entry of the node in the object stack.

3.5 Algorithm for Boolean Operations

This procedure combines the left subtree intersection points and right subtree intersection points according to the boolean operation.

The parameters required are boolean operation, intersection points in left subtree and right subtree. The output is the combined intersection points depending on the boolean operation.

Let LST1 and LST2 be the two stacks containing left subtree and right subtree intersection points respectively and LST is the resultant stack after boolean operation.

The algorithm is then as follows:

a] If left stack is empty then

Case boolean operation of

i) Union : LST ← right stack (LST2)

ii) Intersection : LST ← null stack

iii) Difference : LST ← null stack

otherwise

b] If right stack is empty then

Case boolean operation of

i) Union : LST ← left stack (LST1)

ii) Intersection : LST ← null stack

iii) Difference : LST ← null stack

otherwise

c] Pop the element out of right and left stack

d] While left stack or right stack is not empty do

I] If left and right intersection points are same then

If left subtree stack is empty then

i) assign right stack value to temporary list

ii) pop the right stack for next value

otherwise

If right subtree stack is empty then

i) assign left stack value to temporary list

ii) pop the left stack for next value

otherwise

i) assign left stack value to temporary list

ii) pop right and left stack for next value

otherwise

II] If left stack value is less than right stack value then

i) assign left stack value to temporary list

ii) pop left stack for next value

else

i) assign right stack value to temporary list

ii) pop right stack for next value

III] Store the temporary list value in the resultant stack (LST) depending on boolean operation

e] If left stack or right stack is not empty then go to step (d).

3.6 Ray-Object Intersection Algorithm

This procedure starts at the top of the tree, descends downwards to the terminal nodes, calculates the intersection points and returns up the tree combining left and right subtrees.

The algorithm is as follows:

a] For each pixel in the window do

b] Set the view direction

c] Initialise the intersection point stack

d] If pointer is not pointing to a primitive then

i) get left subtree intersection stack

ii) get right subtree intersection stack

iii) merge the two stacks according to boolean operation.

else

i) transform the ray to primitive's local coordinate system by primitive's world-to-object transformation

ii) do case primitive type of

BOX : Solve 6 ray-plane intersection test

WEDGE : Solve 5 ray-plane intersection test

CYLINDER : Solve 2 ray-plane intersection and one ray superquadric surface intersection test

CONE : Solve one ray-plane and one ray-superquadric surface intersection test

SPHERE : Solve one ray-superquadric intersection test.

3.7 Bairstow's Method [6]

For ray-superquadric surface intersection we have to solve a polynomial for the intersection points. For solving the polynomial Bairstow's method is implemented. This procedure breaks the polynomial in quadratic parts and then calculates the roots of the quadratic part if any.

The algorithm is as follows.

Let a_i contains the coefficients of polynomial and n is the total number of terms in the polynomial (1+ degree of the polynomial).

a] $U \leftarrow a_{n-1}/a_{n-2}$; $V \leftarrow a_n/a_{n-2}$

b] Set

$b_1 \leftarrow 1.0$; $b_2 \leftarrow a_2 - U$

$c_1 \leftarrow 1.0$; $c_2 \leftarrow b_2 - U$

c] Calculate $b_k \leftarrow a_k - a_{k-1} * U - a_{k-2} * V$

d] Calculate $c_k \leftarrow b_k - c_{k-1} * U - c_{k-2} * V$

e] Calculate $Deno \leftarrow c_{n-1} * c_{n-3} - c_{n-2}^2$

$DU \leftarrow (b_n c_{n-3} - b_{n-1} c_{n-2})/Deno$

$DV \leftarrow (c_{n-1} b_{n-1} - b_n c_{n-2})/Deno$

$ERROR \leftarrow ABS(DU + DV)$

f] If Error is greater than or equal to converging limit then go to (b).

g] Calculate root of quadratic part

h] If the remaining part of the polynomial is not quadratic or linear then go to (a).

i] Calculate the root of the quadratic part or linear part as the case may be.

j] Exit.

CHAPTER 4

SOFTWARE DEVELOPMENT

4.1 Implementation Details

The present software has been developed in PASCAL language and implemented on ND 560/CX system. For graphic displays Tektronic 4109 and 4107 raster graphic terminals are used. For this software various local commands of the terminal are interfaced for different graphic procedures.

4.2 Program Structure

Figure 4.1 gives block diagram of the software developed. Initiate data procedure initiates certain data such as it sets number of objects created to zero, selects a default window etc.

Initiate device procedure activates the terminal, clears the graphic displays present on the screen.

The following section give details of the various interactive features.

4.3 Interactive Features

The basic requirement of any geometric modeling system is that it should be highly interactive and user's friendly. The user should have maximum possible freedom for modeling as well as viewing the object. Keeping this as a guide line,

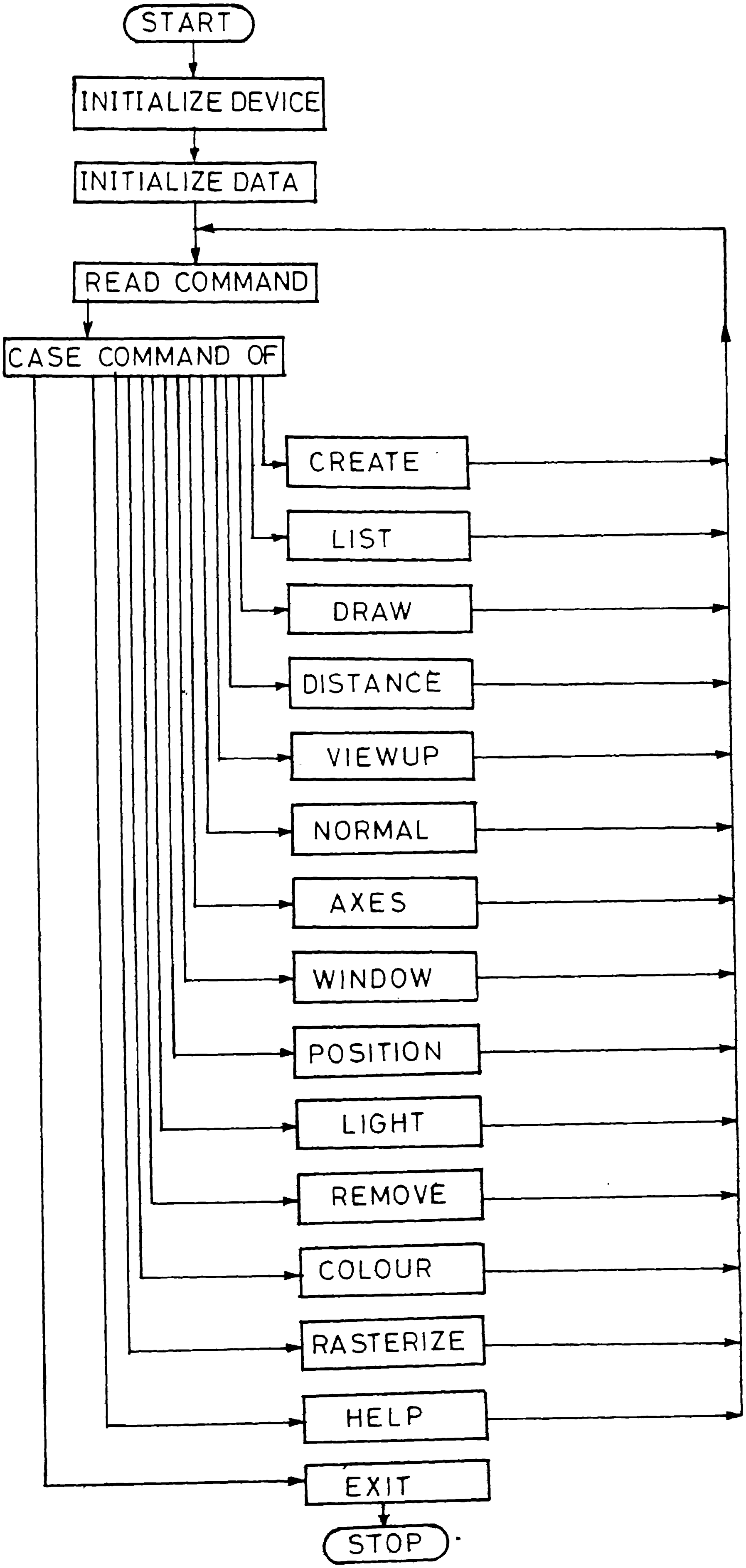


FIG. 4.1 PROGRAM STRUCTURE

certain commands are developed which are simple and highly interactive.

When executed the program displays the message and the prompt

TYPE HELP (FOR HELP)

⇒

'⇒' This prompt is very important because it indicates that the program is waiting for the next command. Whenever a particular command is over, '⇒' is displayed.

4.3.1 <u>Create</u>: This command is most important. With this command only, the user creates different objects by combining two already existing objects or by combining a object and a primitive or by combining two primitives. User can modify an existing object by translating/rotating it.

The primitives along with the associated parameter are:

i) BOX(L, B, H, T, P, W, X, Y, Z, D, S, E, C):

L = Dimension along X direction (an integer)

Default = 100

B = Dimension along Y direction (an integer)

Default = 100

H = Dimension along Z direction (an integer)

Default = 100

T = Rotation about Z axis in degrees (an integer)

Default = 0

P = Rotation about Y axis in degrees (an integer)

Default = 0

W = Rotation about X axis in degrees (an integer)

Default = 0

X = Translation along X axis (an integer)

Default = 0

Y = Translation along Y axis (an integer)

Default = 0

Z = Translation along Z axis (an integer)

Default = 0

D = Object light diffusivity (an integer)

Default = 70

S = Object light specularity (an integer)

Default = 30

E = Object specular exponent (an integer)

Default = 2

C = Colour for the line for drawing the wire frame of object

Default = 15

ii) WEDGE(L, B, H, T, P, W, X, Y, Z, D, S, E, C)

All the parameters are same as discussed for BOX.

iii) CYLINDER(N, A, B, H, T, P, W, X, Y, Z, D, S, E, C)

N = Superquadric degree (an integer)

Default = 2

A = Major axis of the cross section (an integer)

B = Minor axis of the cross section (an integer)

H = Height of the cylinder

Rest of the parameters are same as given for the BOX.

iv) CONE(N, A, B, H, T, P, W, X, Y, Z, D, S, E, C)

All the parameters are same as discussed for cylinder.

v) SPHERE(N, R, T, P, W, X, Y, Z, D, S, E, C)

R = Radius of sphere (an integer)

Default = 100

Rest of the parameters are same as given for cylinder.

The characters used for boolean operations are

'+' for Union

'-' for Difference

'*' for Intersection.

Different cases elaborating how to give the input for create routine are given below:

a) Input for combining two premitives

⇒ CREATE PIPE = CYLINDER (A = 40, B = 40, H = 100) -
CYLINDER (A = 30, B = 30, H = 100)

By this we are creating an object with name attribute as pipe. This consists of two cylinders (parameters are listed) with the second substracted from the first. Those parameters whose values are not read are taken as default.

b) Input for modifying an existing object

A typical command is

⇒ CREATE A = B(T, P, W, X, Y, Z);

Here B is an already existing object which we want to modify. A stores the modified definition of B. The

parameters which can be modified are listed.

c) Input for combining two already existing objects

⇒ CREATE A = B(T, P, W, X, Y, Z) + C(T, P, W, X, Y, Z)

Here B and C are already existing objects. A stores the combination of B and C. Any of the parameters listed can be changed according to requirement.

4.3.2 List: This command gives all the relevant information regarding an object. It gives the list of all the primitives with their parameters and associated boolean operations which are used to model the solid.

A typical command is

⇒ LIST OBJECT-NAME

where OBJECT-NAME in general is the name of the object created. For illustration consider the example given in Section 4.3.1. For listing the command is

⇒ LIST PIPE

The response is as follows:

CYLINDER (2)(0, 0, 0)(0, 0, 0)(40.00, 40.00, 100.00)(2, 70, 30)

-

CYLINDER (2)(0, 0, 0)(0, 0, 0)(30.00, 30.00, 100.00)(2, 70, 30)

This feedback is very essential while modeling a complex object.

4.3.3 Draw: This command draws the wireframe picture of the object. It gives the user graphical feedback regarding the problem during modeling. User can select different colours for different primitives for clear perception.

The command is

⇒ DRAW OBJECT

This will draw all the primitives associated with OBJECT.

4.3.4 Distance: This command specifies the position of view plane with respect to an object or scene. The distance should always be a positive integer as according to the convention followed, the scene is always in front of viewer. A typical command is

⇒ DISTANCE 20

This sets the view plane along the view plane normal at a distance of 20 units.

4.3.5 Viewup: This command specifies the viewup vector required for camera setting. The elements of the vector are real numbers. A typical command is

⇒ VIEWUP 0 0 1

This sets the camera angle in the view plane.

4.3.6 Normal: This command specifies the view normal direction. The vector elements to be keyed in must be integers. A typical command is

⇒ NORMAL -1 -1 -1

This sets the viewer's direction in the global coordinate system.

4.3.7 Position: This command sets the view reference point. All the coordinates to be specified are integers.

A typical command is

⇒ POSITION X coordinate Y coordinate Z coordinate

The default values are 0 0 0.

4.3.8 Axes: This command draws the global coordinate system when invoked. This command should be invoked only after setting the desired camera parameters. The command is

⇒ AXES

4.3.9 Window: This command reads the window parameters which are in screen coordinate system. This helps to select different windows according to this requirements. A typical command is

⇒ WINDOW X min Y min X max Y max

All the window parameters are positive integers.

4.3.10 Light: This command sets the light source in the world coordinate system. The typical command is

⇒ LIGHT X coordinate Y coordinate Z coordinate

4.3.11 Remove: This command removes the light source existing; thus giving the user the flexibility to again place the source of light at desired location. The command is

⇒ REMOVE 1

4.3.12 Colour: This command reads the hue value and accordingly sets the colour indices. These indices are subsequently used for shading the object. Typical command is

⇒ COLOUR VALUE

4.3.13 Rasterize: This is one of the important commands. When invoked this produces the solid model of the object in the window selected. All the operations like setting the camera, creating the desired object, selecting the proper window, setting the light source should be done before this command. A typical example is

⇒ RASTERIZE OBJECT

If the object is not defined or the name attribute keyed in is incorrect then a error is prompted

UNDEFINED OBJECT! TRY AGAIN.

4.3.14 Help: As the name suggests, when this is invoked, a list of all the interactive commands available is provided to the user. To invoke help

⇒ HELP

4.3.15 Exit: To stop the program this command is to be invoked. It allows the control to come out of the interactive repeat loop and stops the entire program. The command is

⇒ EXIT

4.4 Sample Displays

Figure 4.2 shows the picture of Box and Wedge. Figure 4.3 shows the picture of superquadric cylinder with superquadric degree 4 and general quadric cylinder. Figure 4.4 shows the picture of superquadric cone with superquadric degree 4 and general cone. Figure 4.5 shows the picture of superquadric

Figure 4.2

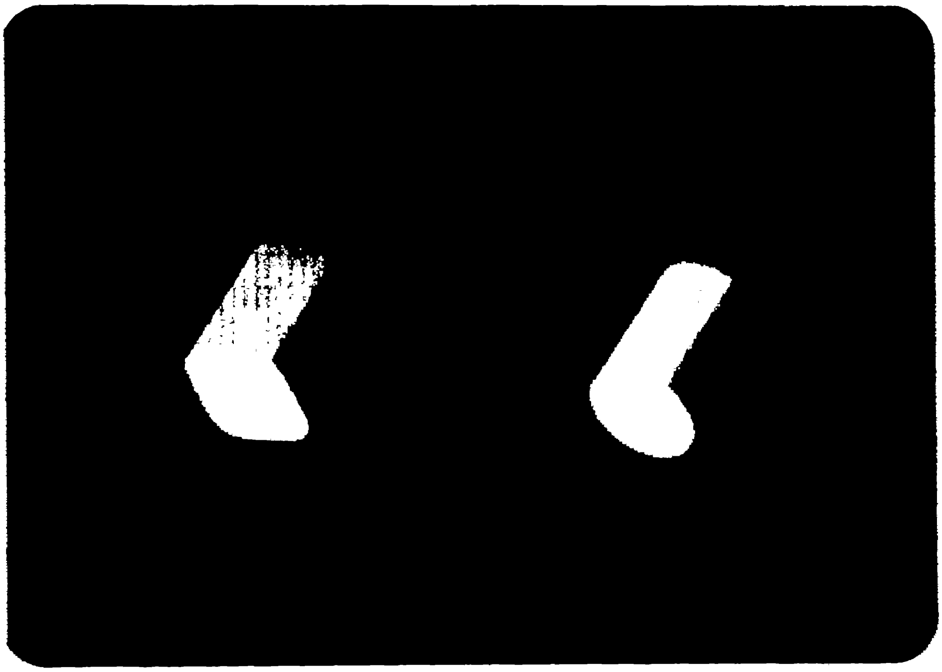


Figure 4.3

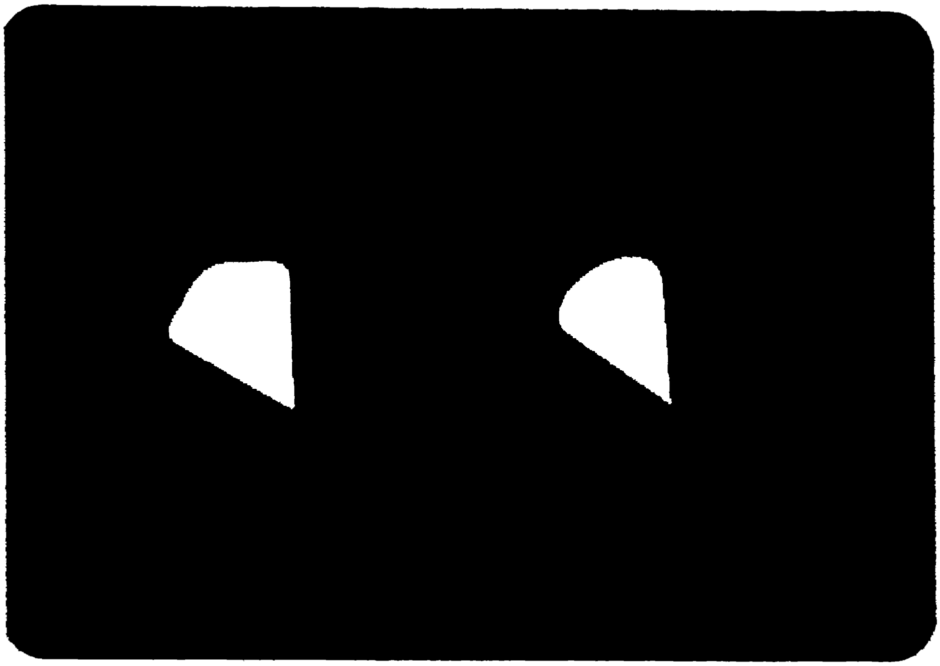


Figure 4.4

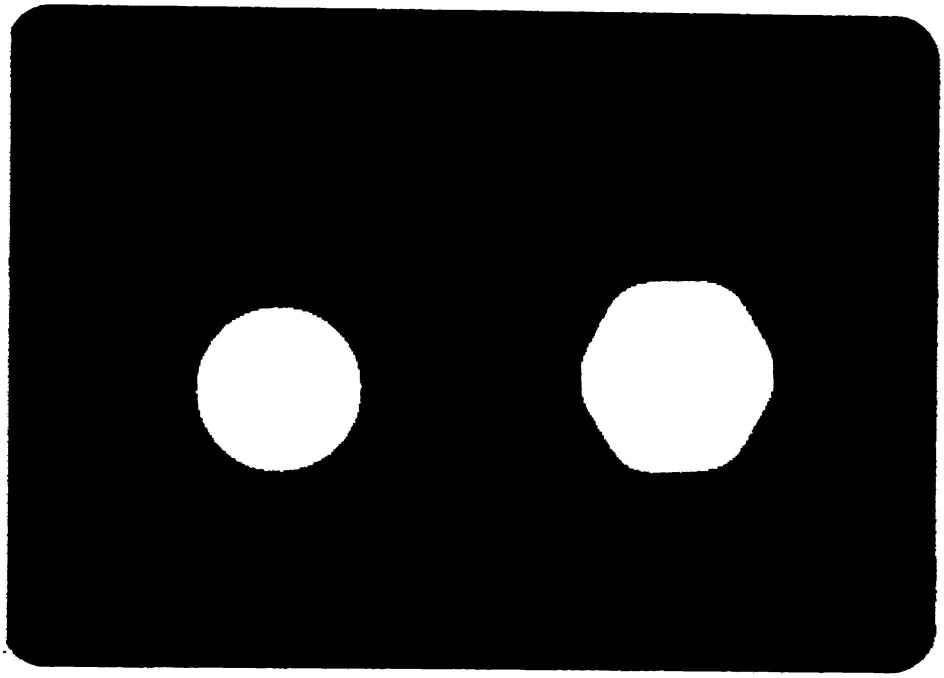


Figure 4.5

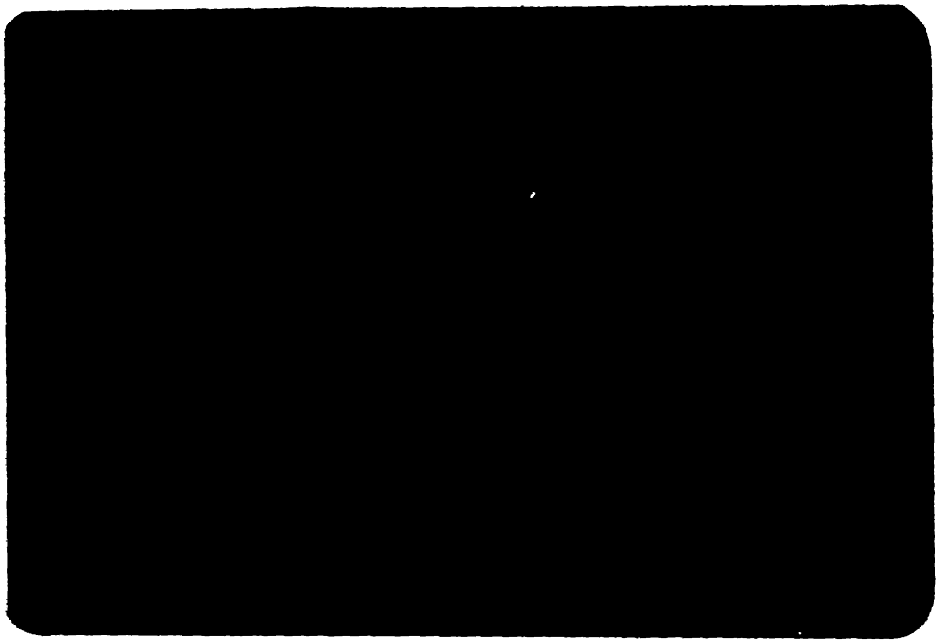


Figure 4.6

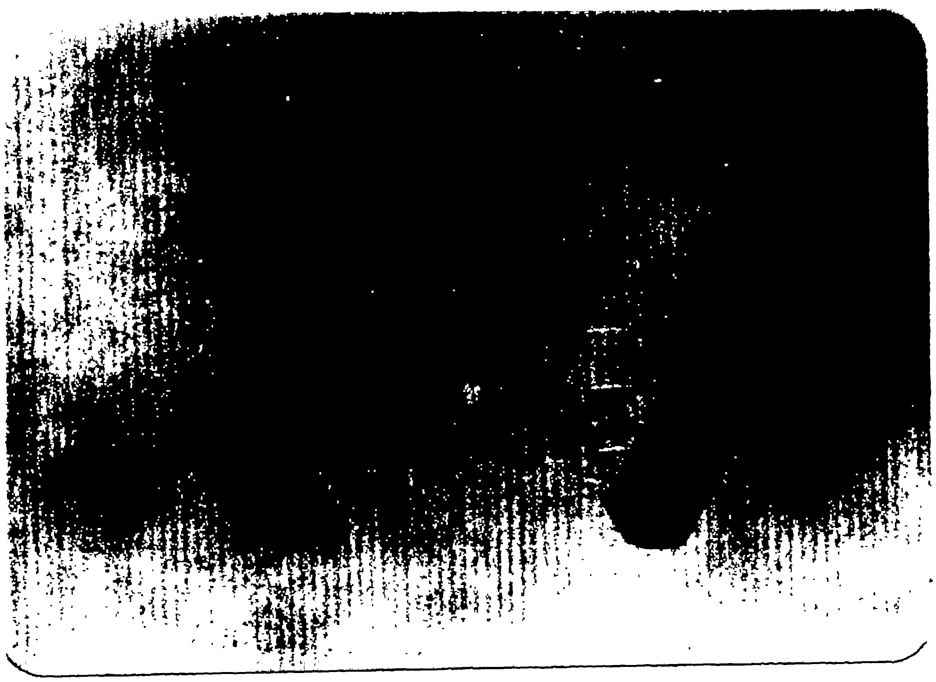


Figure 4.7

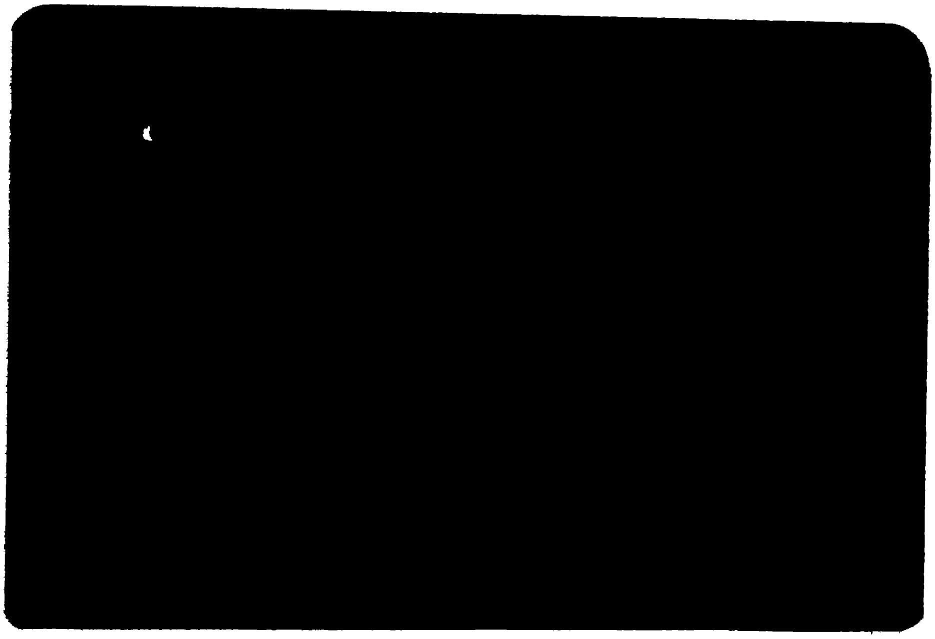


Figure 4.8

sphere with superquadric degree 4 and general sphere. All above are the basic building blocks (primitives) incorporated in the present work.

4.4.1 <u>Union</u>: Figure 4.6 illustrates the union operation. The basic primitives used are box and superquadric cylinder. With reference to Section 4.3, for producing this object the input was given in the following manner:

```
⇒ DISTANCE 20
⇒ VIEWUP 0 0 1
⇒ NORMAL -1 -1 -1
⇒ LIGHT  100  -200  -400
⇒ CREATE OBJECT1 = BOX(L = 60, B = 60, H = 10);
⇒ CREATE OBJECT2 = CYLINDER(N = 4, A = 20, B = 20, H = 60,
  Z = -30, X = 30, Y = 30);
⇒ CREATE
⇒ WINDOW 100  100  200  200
⇒ RASTER OBJECT1
⇒ WINDOW 300  100  400  200
⇒ RASTER OBJECT2
⇒ WINDOW 200  200  300  300
⇒ RASTER OBJECT3
```

4.4.2 <u>Difference</u>: Figure 4.7 illustrates the difference operation. The basic primitives used are box and superquadric cylinder.

With reference to Section 4.3, for producing this picture input was given in following manner:

```
⇒ DISTANCE 20
⇒ VIEWUP 0 0 1
⇒ NORMAL -1 -1 -1
⇒ LIGHT  100  -200  -400
⇒ CREATE OBJECT1 = BOX(L = 60, B = 60, H = 10);
⇒ CREATE OBJECT2 = CYLINDER(N = 4, A = 20, B = 20, H = 60,
  Z = -30, X = 30, Y = 30);
⇒ CREATE OBJECT3 = OBJECT1( ) - OBJECT2( )
⇒ WINDOW  100  100  200  200
⇒ RASTER OBJECT1
⇒ WINDOW  300  100  400  200
⇒ RASTER OBJECT2
⇒ WINDOW  200  200  300  300
⇒ RASTER OBJECT3
```

4.4.3 Intersection: Figure 4.8 illustrates the intersection operation. The basic primitives are the same i.e. box and superquadric cylinder. With reference to Section 4.3 for producing this object the input was given in this manner:

```
⇒ DISTANCE 20
⇒ VIEWUP 0 0 1
⇒ NORMAL -1 -1 -1
⇒ LIGHT  100  -200  -400
⇒ CREATE OBJECT1 = BOX(L = 60, B = 60, H = 10);
⇒ CREATE OBJECT2 = CYLINDER(N = 4, A = 20, B = 20, H = 60,
  Z = -30, X = 30, Y = 30);
⇒ CREATE OBJECT3 = OBJECT1( ) * OBJECT2( )
⇒ WINDOW  100  100  200  200
```

```
⇒ RASTER OBJECT1
⇒ WINDOW  300  100  400  200
⇒ RASTER OBJECT2
⇒ WINDOW  200  200  300  300
⇒ RASTER OBJECT3
```

CHAPTER 5

SUMMARY

5.1 Conclusions

A geometric modeler based on constructive solid geometry representation scheme using superquadric solids as primitives has been implemented and analysed.

Many times in engineering practice it is observed that the workpieces do not have sharp edges. These classes of workpieces are difficult to model using generally used prism and quadric solid primitives. It is observed that the use of the superquadric solids gives the user greater flexibility in modeling such objects. These superquadric solids cover a wide range of objects which are very useful in modeling different complex objects. It is possible to modify the basic primitive itself by changing the superquadric degree according to the requirement.

5.2 Suggestions for Further Work

a) Ray tracing in general qualifies as a brute force method for solving problems. Memory and CPU usage is directly proportional to the scene complexity.

Most of the efforts are concentrated on calculating ray-solid intersection points. Even though there is no object in the path of the ray still it visits each node of the tree

and performs all the intersection tests. By using a minimum bounding box around a solid in the composition tree, the exhaustive search for a ray-solid intersection then resembles an efficient binary search. This improves the efficiency of the ray-tracing procedure to great extent. So bounding box technique can be incorporated with present work to improve the efficiency of ray-solid intersection procedure.

b) Various application subsystems for calculating mass, moment of inertia volume which are essential for a solid modeler can be interfaced with the present work. This would result in a complete solid modeler.

c) In the present approach, the complex object is created by combining different primitives through certain operations. The same approach can be applied in reverse manner also. Given a complex object composition one can get back the basic primitives used to model the object. This will be very helpful for deassembling operation of the mechanical components.

REFERENCES

1 A.A.G. Requicha and H.B. Voelcker, "Solid modeling: A historical summary and contemporary assessment", IEEE Computer Graphics and Applications, March 1982.

2 A.A.G. Requicha, "Representations for rigid solids, theory methods and systems", ACM Computing Survey, 12, 4 Dec. 1980.

3 I.D. Faux and M.J. Pratt, "Computational geometry for design and manufacture", Ellis Horwood Ltd., Chichester, 1979.

4 N. Okino, Y. Kakazu and H. Kubo, "Theories for graphics processors in TIPS-1", Computers and Graphics, Vol. 7, No. 3-4, pp. 243-258, 1983.

5 S.D. Roth, "Ray casting for modeling solids", Computer Graphics and Image Processing, 18, pp. 109-144, 1982.

6 F.B. Hildebrand, "Introduction to numerical analysis", Tata McGraw-Hill, 1984.

7 Steven Harrington, "Computer Graphics: A programming approach", McGraw-Hill, 1986.

8 David F. Rogers, "Procedural elements for computer graphics", McGraw-Hill, 1985.

9 John Sandor, "Octree data structure and perspective imagery", Computers and Graphics, Pergamon Press, Vol. 9, No. 4, pp. 393-405, 1985.

APPENDIX A

TRANSFORMATION MATRICES

A1 Object-to-Global Transformation Matrix

Let SX, SY and SZ be the scaling factor in X, Y and Z direction. θ, ϕ and w are the angle of rotations in radians about Z, Y, X axis. X, Y, Z be the translations along X, Y, Z axis then the transformation matrix [T] is given by

$$\begin{bmatrix} \cos\theta.\cos\phi.SX & SX(\sin\theta.\cos w - \cos\theta.\sin\phi.\sin w) & SX(\sin\theta.\sin w + \cos\theta.\sin\phi.\cos w) & 0 \\ -\sin\theta.\cos\phi.SY & SY(\cos\theta.\cos w + \sin\theta.\sin\phi.\sin w) & SY(\cos\theta.\sin w - \sin\theta.\sin\phi.\cos w) & 0 \\ -SZ.\sin\phi & -SZ.\cos\phi.\sin w & SZ.\cos\phi.\cos w & 0 \\ X & Y & Z & 1 \end{bmatrix}$$

The inverse of the matrix [T] gives global-to-object transformation matrix.

A2 Camera-to-World Transformation Matrix

The purpose of this transformation is to establish relationship between camera and world coordinate system.

Let DXN, DYN, DZN be the component of view normal vector. DIS be the viewer's distance. XUP, YUP, ZUP be the component of view up vector. XR, YR, ZR be the coordinates of view reference point. Then the transformation matrix [M] is given by

$$M = TR_xR_yR_z$$

where

$$[T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -(XR+DXN*DIS) & -(YR+DYN*DIS) & -(ZR+DZN*DIS) & 1 \end{bmatrix}$$

$$[R_x] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -DZN/V & -DYN/V & 0 \\ 0 & DYN/V & -DZN/V & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where

$$V = (DYN^2 + DZN^2)^{1/2}$$

$$[R_y] = \begin{bmatrix} V & 0 & -DXN & 0 \\ 0 & 1 & 0 & 0 \\ DXN & 0 & V & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[R_z] = \begin{bmatrix} YUP/RUP & XUP/RUP & 0 & 0 \\ -XUP/RUP & YUP/RUP & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$RUP = (XUP^2 + YUP^2)^{1/2}$$